Diploma Thesis

Graphical Authoring Tools for Web Content Management

Simon Zweifel, D-INFK
simon_zweifel@yahoo.com

February 27th 2002

Institute of Information Systems
Swiss Federal Institute of Technology (ETHZ)

Diploma Professor:
Prof. Moira C. Norrie

Supervisor:
Michael Grossniklaus

Abstract

In website development, Content Management Systems are being used more and more every day. A lot of different products exist already. One of these is CMServer, a Content Management Server, based on OMS and developed at the Global Information Systems Group at ETH Zurich.

In this diploma thesis we present a tool, which simplifies drastically the usage of CMServer by providing the user with a simple and intuitive interface.

Contents

1 Introduction 7
1.1 Features of common web authoring tools 7
1.1.1 Content Editing 7
1.1.2 Visual Editing and Web Page Layout 8
1.1.3 Site-wide Features & Tools 8
1.2 Features of Content Management Systems 8
1.2.1 Separation of Content, Structure and Layout 8
1.2.2 Workflow-support 9
1.2.3 Multilingual Content 9
1.2.4 Object reusability 9
1.3 XWCM 9

2 Requirements 11
2.1 General requirements to a Graphical User Interface 11
2.2 Need of a new Tool 11
2.3 Requirements to the XWCM – GUI 12
2.3.1 Model-Generation 12
2.3.2 Object-Creation 12
2.3.3 Site Structure 13
2.3.4 Presentation 13
2.3.5 Compatibility 13

3 Design 15
3.1 Design Goals 15
3.2 Basic Steps 15
3.3 Preferences 16

4 Implementation 17
4.1 Principles 17
4.2 Packages 18
4.2.1 ch.ethz.globis.cmserver.gui 18
4.2.2 ch.ethz.globis.cmserver.gui.database 19
4.2.3 ch.ethz.globis.cmserver.gui.model 21
4.2.4 ch.ethz.globis.cmserver.gui.objects 21
4.2.5 ch.ethz.globis.cmserver.gui.structure 21
4.2.6 ch.ethz.globis.cmserver.gui.layout 22
4.2.7 ch.ethz.globis.cmserver.gui.tests 22
4.2.8 ch.ethz.globis.cmserver.gui.images 23
4.2.9 ch.ethz.globis.cmserver.gui.exception 23

4.3 Classes 24
4.3.1 GUICollection 24
4.3.2 GUIComponentType 25
4.3.3 GUIObject 26
4.3.4 GUIDirectory 27
4.3.5 GUIContent 27
4.3.6 GUITestsPanel 28
4.3.7 GUIPrefs 28
5 Conclusions 29
5.1 Fulfillment of Requirements 29
5.2 Missing Features 30
5.3 Future Work 31
5.4 Minimal database-requirements 31

1 Introduction

The Internet is growing rapidly and the competition in the web is getting bigger every day. It becomes more and more important to provide top-actual information on your own website to remain competitive. That's why "usual" tools like "Microsoft FrontPage" or "Adobe GoLive!" often don't fulfill the requirements of webmasters anymore. In the past years content management systems have become more important. These systems allow a much easier administration of information for a website.

Such a system has been developed at the Global Information Systems Group [3] at the Institute of Information Systems at ETH Zurich. It's called "Extensible Web Content Management System" (XWCM) [1]. However, using this system in practice can be quite complicated, as there exists no simple user interface for the data management. The implementation of a graphical user interface for XWCM was the objective of this diploma thesis.

In this first chapter we're going to have a look at some existing web authoring tools and their functionality. At the end of the chapter we will explain some of the features of the XWCM system.

1.1 Features of common web authoring tools

When somebody has been building a web page in the last couple of years, he was most certainly creating a lot of HTML-pages. The easiest and most common way to do this is to use one of some commercial products that provide intuitive HTML-editors. The most famous ones are Microsoft FrontPage and Adobe GoLive! Here is a short overview of their features, which are partly also of interest for our user interface.

1.1.1 Content Editing

- The main and most comfortable feature of the web authoring tools are WYSIWIG editors that provide the user with features that are known from tools like Word, etc… Formatting text becomes very easy and for first time users it's not a problem if they don't know anything about HTML.
- The possibility to be able to look at the "raw" HTML is very useful, especially for more advanced users.
- Certain objects like URLs, fonts or colors can be "reused". All used objects are managed in lists, where the Webmaster can easily find them when he needs them again.

1.1.2 Visual Editing and Web Page Layout

- The visual layout, the HTML source and previews of web pages are accessible through tabs or through specific menus.
- A special toolbox holds all the known object types like text, image, table, layers, buttons, etc… They can be positioned precisely on a web page through drag-and-drop.
- An attribute inspector lists all the properties of objects. There one can edit them right in place. This inspector is either accessible through a right mouse button click or in a separate, context sensitive window.
- The support for more complex features like frames, forms, javascript, applets, images, audio is a feature that depends strongly on the used version of HTML. This makes it necessary to release new versions of the application each time there are changes in the HTML-standards.

1.1.3 Site-wide Features & Tools

- The whole site structure can be viewed in a tree, containing the directories and pages. This view is easy for users to understand, as it doesn't differ much of the file-view they know from the computer's file system.
- Objects can be reused by saving them into components. This feature can be used for example for navigation bars or page-headers.
- Tools like FTP support and file synchronization simplify the updating of the published site with the locally stored one.
- Lists for broken links, orphaned pages and several reports are useful for managing big websites. Very often the user can also configure reports, so that they fit his needs.
- Help for the most common problems is accessible through tutorials, wizards or help pages.

While most of these features are almost standard nowadays in web authoring tools, not all of them are of the same importance. In chapter 2 we will give a listing of the requirements for our content management system. It is evident, that not all of the mentioned features have to be implemented in this first version.

1.2 Features of Content Management Systems

In this chapter we will give a very compact overview of the main properties of Content Management Systems, as they were defined in [1]. For a deeper discussion please read the mentioned thesis of Michael Grossniklaus.

1.2.1 Separation of Content, Structure and Layout

The most important property of content management systems is the separation of content, structure and layout. With a strict separation, new content can easily be added or removed to a website without having to define new HTML or change existing pages.

The separation of the layout makes it possible to have a website that can be shown in different encodings without changing anything on the structure of the site. For example the same

content can be shown in HTML for web browsers, or one can get the contents in WML, used for wireless access to the Internet.

1.2.2 Workflow-support

Because websites grow very fast, there is often more than one person who controls the development. With integrated workflow-support, the ability to control the different states of pages is improved a lot.

Additionally so-called gatekeepers assure, that pages are just accessible when they are allowed to. Good examples can be a state called “published” or a gatekeeper that shows the page just, if a certain date has passed.

1.2.3 Multilingual Content

A very nice feature of content management systems is the ability to define the same content in several different languages. Most objects can simply be associated with several contents. When constructing the requested page, the content of the appropriate language is fetched from the database. Thanks to the separation of content and layout it is no big deal anymore to translate a whole site into another language.

1.2.4 Object reusability

Quite often specific objects are used on several different pages on a website. This can be a navigation-bar or a picture like a logo, but also some text- and other objects. Reusability of objects means that these items are just stored once and then referenced from the different places where they are needed. Instead of putting a navigation-bar on each page, one single navigation-bar is created and included on each page. However, this single object will look different, depending on the context.

1.3 XWCM

XWCM is a Java Servlet, which uses data from an OMS database. OMS is an object-oriented database model that has also been developed at the Global Information Systems Group at ETH Zurich [3].

Storing data into an OMS database means dealing with types, objects, collections and associations. XWCM consists of a small set of predefined types. They define the core functions and structure of a website. For example there is a type for picture-objects or one for link-objects. The user can create new subtypes or can use the existing ones to create his own website.

XWCM implements all of the above-mentioned features of Content Management Systems. Whenever a visitor to an XWCM-site requests a page (e.g. through a browser), all the needed objects for the page are fetched from the database. In a second step an XML-representation of the page is created. This includes all the attribute-values in the right language. The last step is then the application of XSL-Templates, which define how the XML shall be translated into the final representation (e.g. HTML).

In XWCM the user can define most objects needed for this process. This includes:

- User-defined types as subtypes of the existing ones
- New languages, as a base for new and existing objects
- New objects, pages and directories
- Workflow states and Gatekeepers
- XSL-Templates for formats like HTML or WML

Especially the combination between defining own types and own XSL-templates makes the whole system very powerful. There are almost no restrictions on the pages that can be created.

2 Requirements

In this chapter we will give an overview of the requested features to the Graphical User Interface, which had to be implemented.

The first part looks at some common requirements of Graphical User Interfaces. After that a substantiation of the needs of a new tool and a listing of the requirements is given.

2.1 General requirements to a Graphical User Interface

For a useful and successful program, some basic features are quite important. Especially the ease-of-use can be crucial when it comes to using a program. If a program abides some common guidelines, the program becomes more useful to the customer.

Look & Feel

A very important requirement is a known Look & Feel. That means, that the program looks and behaves like most programs do on the same computer. The user is used to these programs and therefore does not need to learn new functions. A simple example could be to label the buttons with “OK”, “Cancel”, “Save” or similar, like it’s done in almost every application.

The general look depends on the operating system. Buttons, windows, menus, dialogs, etc… look different depending on the used platform. Therefore a program that shall be available on different platforms should also use the corresponding Look & Feel.

Usability

Another requirement is, that the application is comfortable to use. This includes especially stability. Nobody will use a program that crashes repeatedly.

The GUI should also be clear. Too many controls or information on one screen confuse most users. The whole functionality should be structured into some main functions and the program should then provide an easy interface to switch easily between them. That way the user does not loose the overview when working on a big amount of data.

2.2 Need of a new Tool

As stated above, the whole data is stored in an OMS-database. There exist some tools for editing such databases. One example is OMS Pro. These tools allow creating, editing and deleting any objects within the database. Unfortunately in the case of XWCM this becomes complicated because the underlying schema is very complex. To be able to create a website, a potential user first has to understand the whole XWCM-schema. Here is a short example:

The following steps are necessary to create a simple picture-object within the database:

1. Create a <cmpicture> object and set all the values (width, border)
2. Add this new object to the right collections (<CMPictures> and <CMComponents>)
3. Create at least one <cmpicturecontent> to get a content-object for the picture
4. Add them to the collections <CMPictureContents> and <CMContents>
5. Find the associations <hasContent> and <hasPictureContent> and add the two new objects to both of these associations
6. Find the correct language-object and add the association (content, language) to the following collection: <inLanguage>

After the picture has been created, it does not yet appear on any page of the website. A lot of further steps have to be taken, until a page is fully set up.

Especially the fact that so many operations have to be completed until the object is fully described made it necessary to implement a tool that is specialized for this database. Most of the above operations could be done automatically. E.g. once the user decides to create a new picture object, it is clear he will also need an appropriate content object which is linked to the picture.

Read more about the goals of this tool in the next chapter.

2.3 Requirements to the XWCM – GUI

The main goal of the new tool was to have an application, which would provide as much functionality as possible, and still would hide most of it. The user should not have any special knowledge about the used database and should still be able to do most of the things he would be able to do with OMS Pro.

In the following chapters we provide an overview over the common tasks the new application had to provide.

2.3.1 Model-Generation

A nice and very useful feature of XWCM is its extensibility. The user can create new types and new objects, such as they fit best his needs.

One goal of the GUI was to hide the database-schema from the user and to provide an easy interface where one could create new types with their attributes or delete existing ones. As there are collections needed for each type, these should be created implicitly. Also the constraint between a collection and its subcollections has to be hidden to the user.

2.3.2 Object-Creation

In the example mentioned in chapter 2.2 it could be seen that the creation of objects is not that easy in XWCM. Especially new, inexperienced users would have great problems to get accustomed to the very complex schema.

With the GUI it should be possible to create new objects with one click. Especially some common tasks such as adding the object to the right collections or adding new content to an object should be done automatically. Also the creation of a new language and corresponding content-objects is needed.

2.3.3 Site Structure

Webmasters that are used to traditional tools as described in chapter 1.1 are accustomed to work with the site structure. The site structure is usually a tree that starts at the root directory and includes all subdirectories and pages. Within the site structure it is very easy to add or remove new pages and directories.

This feature is also necessary for the new tool, as the site structure is part of the concept of XWCM. Both the sitemap-object and the navigation-object also use the structure of the site.

2.3.4 Presentation

For the presentation of the pages and objects, XSL-templates are needed. Defining such templates in OMS Pro is not easy, as the XML representation of the objects is not known. One basic requirement to the new GUI is that the XML should be viewable. Each object has an XML representation that should be displayed, when the XSL-templates are to be created.

A last requirement to the GUI is the ability to preview the objects after they have been assigned a template, to test the template.

2.3.5 Compatibility

As XWCM is still under construction and will be rebuilt in the near future, one goal while creating a GUI was to remain as general as possible, such that future versions might also work with this GUI. This includes that the application should not be fixed on the database schema.

A second compatibility requirement is the compatibility with other databases. The new application should be able to open valid databases that have been developed using other tools. This way we can ensure, that even the functions, which are not available in the GUI, can be used by editing a database in another editor like OMS Pro.

3 Design

The previously described requirements to the GUI were the starting point in designing the overall look of the program.

3.1 Design Goals

One premier goal was to have an easy program, which helps the user in creating or editing his websites. It was important to provide an interface that supports at its best the usual workflow of a webmaster. Furthermore, all the main functions should be available at any time.

Another goal was to keep it as simple as possible. Beside the fact, that no superfluous information about the database should be visible, the program had to be convenient, even for new inexperienced users.

3.2 Basic Steps

The development of a website based on XWCM has been divided into five major steps, which are also reflected in the design of the user interface. Each step can easily be accessed through tabs on the top of the window or through menus. The first four steps correspond to the requirements (chapter 2.3.1 - 2.3.4).

Model

In the first step the database model can be edited. But instead of defining types, collections and associations, the user has only the possibility to define subtypes of <cmcomponent> with their attributes. All other objects are created implicitly and are not visible for the webmaster.

Objects

In the Objects part of the application the webmaster creates objects and their content. The advantage of this design is that the user does not have to know about the different objects used in the database. The system implicitly creates an object, the required content-objects and the appropriate entries in the right associations. In the end however, the webmaster simply sees an object with different properties depending on the language.

Structure

The structure of the whole site, including all the pages and directories that are available through the browser is designed in the third step. Additionally so-called containers are also included. They are very important components for structuring a website and for reusing objects.

Existing objects can then be added to these pages or containers. The fact that pages are a special kind of containers, which are simply stored in another collection, is hidden to the webmaster.

Layout

The fourth step in creating websites with XWCM is to create the templates, which define, how the pages will be encoded. The webmaster can create new templates, or edit existing ones. Then he can assign them to the objects, which he used in the site-structure. To help him creating correct templates, the program shows the XML representation of each object when it is selected.

The possibility to get a preview of the final page simplifies this task, because the user can see directly, how the different templates affect the resulting page.

Tests

The last step is a collection of tests that can be used to validate the database. The basic idea is that by providing some tools that are easy to use, the webmaster can test if his database fits the requirements of the underlying XWCM System.

This part is easily extensible, such that further improvements to the program will include more tests. Additionally to the tests, reports or other automatic results from operations could be included into this part of the program.

3.3 Preferences

The only feature that is not covered by the basic steps is the management of the languages. It is assumed that most webmasters will create the languages once and then don't have to access them anymore. To save some space in the tabs and improve the overall clarity, the language options have been moved to the site-properties, which are accessible through a menu.

The preferences, which do not depend on the underlying database, are also accessible through another menu. This includes the choice of a different Look & Feel and the settings for the layout preview that needs to know about the user's preferred language.

A special case is the handling of different template-types. Within XWCM different template-types are supported. In the first version, these were HTMLTemplates and WMLTemplates. To keep it transparent, the graphical user interface is always based on one of these types. The webmaster can choose one from the preferences, and then only the templates of the correct type are shown. However, the templates of other types are not lost, they are simply hidden.

To read more about how to use the program, please read the User Manual [4].

4 Implementation

CMSGui is the first implementation of a GUI for XWCM. It has been implemented in Java, using the Swing libraries. The main advantage of using Java is that the program is usable on several different systems. And the program can always have the look and feel of the underlying operating system!

A second advantage of using Java is the already existing OMS Java, an API to access OMS databases from Java. Although it still needs to be improved it provides an easy interface to read and edit such databases.

This chapter explains what the ideas were while implementing CMSGui. Chapter 4.1 describes the main principles used, especially to make it as compatible as possible to future releases of XWCM. In the next chapter all packages are listed and described shortly. In the last chapter (0) the most important classes are explained in detail. The main goal is to make it understandable, what the main function of these classes is, and how they work together. This chapter is especially written for people who want to extend CMSGui. It should simplify the understanding of the source code. It is strongly recommended to first read this chapter before editing the source code.

4.1 Principles

The underlying XWCM is still under construction. Therefore CMSGui had to be implemented as general as possible, such that changes in the future would not affect the tool too much. Still changes in the overall architecture of XWCM will probably also require changes in the GUI.

This means particularly, that the new tool should not depend too much on names, types or associations. For example, when CMSGui was developed, two sorts of templates were defined in the database: HTMLTemplates and WMLTemplates. Certainly more template-formats are planned for the future. So CMSGui does not know about these two formats in particular. Instead it looks in the database for a type called “CMTemplate” and then finds all subtypes of it.

Another important principle was to make it compatible with other database-editors. Databases, which had been created with OMS Pro, for example, should also be readable with CMSGui. A short example: At some places the GUI allows just one strict operation. But it has been built, such that it would also open databases, which were not created that strictly. For example, when creating a new type, the GUI implicitly creates exactly one corresponding collection. There is no way to create more than one collection for a type in CMSGui. But when one opens a database that has several collections of one type, they are also displayed.

4.2 Packages

The following descriptions do not explain how the program can be used, but rather how it is implemented. To understand the program from a users perspective, please read the Users Manual [4].

CMSGui is part of the package "ch.ethz.globis.cmserver", which contains also CMServer, the implementation of XWCM. For instructions about the installation and starting of CMSGui, read the Manual.

The java-classes have been grouped into some packages, depending on their functionality. There are packages for each of the five panels that appear in CMSGui (chapter 4.2.3 - 4.2.7). One important package includes all the database objects that are displayed within CMSGui and therefore need a GUI implementation (chapter 4.2.2).

4.2.1 ch.ethz.globis.cmserver.gui

This package contains the main classes of CMSGui. It includes all the different windows of the program and some helper-classes.

CMSGui.java	Main class Instantiates all the needed windows in the right order (e.g. Log Window has to be created first, as it is used by all the others)
GUIMain.java	Main window Creates first the menubar and handles its events, such as opening or closing databases. This window is also used as the base window for all the dialogs or file choosers used later on.
GUILog.java	Log window Provides some static interfaces to write data into the log window or on STDOUT.
GUIPrefs.java	Preferences window This class manages the preferences for the whole program. This includes the writing and reading of the preferences to a file and the graphical interface to edit these. Part of the preferences is also the list of recently opened items.
GUIProperties.java	Properties window Contains the properties of the opened website. This includes a list of the languages defined within a site. Like GUIPrefs it is both a graphical interface to edit these properties and an API to access the data.
GUIMenuBar.java	Menu Bar The menu bar is part of the main window. It consists of several items. Some of them are implemented within this class; others are defined in their own classes (e.g.: The menu-item corresponding to the log-window is handled in the GUILog-class). GUIMenuBar collects all the menu-items and groups them into the final menus.

GUIProgressBar.java	Progress Bar An easy implementation of a progressbar that displays some information text below. GUIProgressBar can be initialized with a maximal value and then can be informed, whenever the value changes.
GUIPaneInterface.java	Abstract interface for the panels used in the main window. It defines methods, which are used when a tab is selected or closed.
GUIConst.java	Collection of constants used in the whole program. This includes a lot of strings that are visible to the user. By having stored them at one single place, changes to the GUI are easier.
GUITools.java	Utilities providing some handy methods that are used at different places throughout the whole program.
SwingWorker.java	SwingWorker is an abstract class for performing GUI-related work in a separate thread. It is used when opening a website to keep the GUI (including the progressbar) responsive. This class is available for free on the Java-website at sun. (java.sun.com)

4.2.2 ch.ethz.globis.cmserver.gui.database

The database package is responsible for all access to the database. Every database-object used in the GUI has a corresponding GUI-object, which is created when this object is first used.

GUIDBAccess.java	All direct access to the OMS database is done through this class. It is the class that opens, saves or closes the database. It is here, that most of the objects are collected and created. All public methods are static, such that it can be accessed from any point in the program.
GUIComponentType.java	GUIComponentType is the GUI implementation of some special types that are used very often. All objects which can appear on a website (including the pages and directories themselves) are of subtypes of <cmcomponent>. These types are called “ComponentTypes” in CMSGui. This class holds the GUI for such a type, where the user can edit its attributes, its name, and so on… It also knows, if the type is not editable and displays a different interface in this case.
GUITypeAttribute.java	This is the GUI of a single attribute of a ComponentType. Here the user can change the name, bulk and type of such an attribute.
GUIEditableObject.java	This abstract class defines the common methods for an object, which can be created, edited and deleted by the user of the program. This includes especially a method, which saves all the changes that occurred.

GUIObject.java	GUIObject represents a database-object that is of some "ComponentType". An object can be created, mutated or deleted. It has to know about the collection, in which it is stored and about the associated content-objects and templates. A GUIObject is created whenever an object is used. It is then stored in a list where it can be found later for re-use.
GUIEditableAttribute.java	An object consists of several EditableAttributes. These are all attributes of the underlying type and their corresponding values. EditableAttributes do automatically display the right interface-component. Strings or Integers can be entered in a textfield, while more complex objects can be chosen from a drop-down menu.
GUIContainer.java	GUIContainer is the implementation for the Containers, which have to be displayed slightly different than "usual" objects. Containers have an associated set of components, which can be ordered. GUIContainer is a subclass of GUIObject.
GUIDirectory.java	Another extension of GUIObject is GUIDirectory. All the specific methods for directories (like creating pages or subdirectories) are implemented here.
GUICollection.java	The graphical interface of a collection is not very complex. It needs just a toString()-method to display its name when the collection is used. But some specific methods within this class help to manage the collections together with their types.
GUIContent.java	GUIContent is a subclass of GUIEditableObject. The displaying of the right contents is part of the corresponding GUIObject. GUIContent just knows what attributes have to be filled with what values.
GUILanguage.java	GUILanguage is a small class for the management of the used languages.
GUITemplate.java	This is the GUI implementation of the templates. Beside the content of a template, the references to this template are also handled through this class. This is necessary for the deletion of templates.
GUITextValue.java	Somewhat special objects in the database are the ones of type "text". They are used for longer strings. Attributes can also be of this type, in which case they are displayed in a text area. But if the attribute is not univariate, a list of all the text objects has to be displayed. This class provides a "short view" of such objects.

4.2.3 ch.ethz.globis.cmserver.gui.model

The model package contains all the classes, which are used to edit the underlying model of the database.

GUIModelPanel.java	This is the panel that is part of the main window. It's an extension of GUIPaneInterface that is used by the main window. Like all other Panels of that window it is displayed when a database is opened and removed again, when the database is closed.
GUITypeTree.java	GUITypeTree is the tree holding all GUIComponentTypes. The root node is the <cmcomponent>-type, which is the parent of all other types. The user has the possibility to create new subtypes or remove existing ones. Depending on the fact if the types are editable or not, they are displayed in a different colour. Events like the selection of an entry are handled by the GUIModelPanel.

4.2.4 ch.ethz.globis.cmserver.gui.objects

The objects package contains just one single class, which is instantiated by the main window.

GUIObjectsPanel.java	Implementation of the Objects panel, which is divided into three parts, the list of types, the list of corresponding objects and the part, where the properties of the selected object are displayed.

4.2.5 ch.ethz.globis.cmserver.gui.structure

The structure panel allows to manipulate the pages and directories of the website. In this package, all the classes that are used for that panel are joint.

GUIStructurePanel.java	This class creates the panel used for the management of the directories and pages. It is divided into a structure tree on the left side and the properties of the selected object on the right side. GUIStructurePanel implements the ActionListeners for the structure tree.
GUIStructureTree.java	Implementation of a tree starting at the root node of the site and including all the subdirectories, pages and containers. One feature of the structure tree is, that containers (including pages, which are special containers) that are used more than once are displayed in italics. To make sure, that changes in the overall structure of the site are reflected correctly in the tree, the subtree of the selected entry is reloaded whenever the user selects another item.

4.2.6 ch.ethz.globis.cmserver.gui.layout

The layout package is the counterpart to the layout panel of CMSGui. This includes the management of the templates.

GUILayoutPanel.java	The implementation of the layout panel is responsible to collect all the templates that match the selected mimetype in the preferences. One speciality of this class is an Actionlistener that is added to the save button of the preferences window. When the user of CMSGui changes the preferences, GUILayoutPanel is notified through this Actionlistener. This way it is assured, that the right templates matching the user's selection are displayed.
GUILayoutTree.java	GUILayoutTree is an extension of the previously described GUIStructureTree (chapter 4.2.5). There are two additional features implemented in the layout tree. The entries of the tree are not just all the directories and containers, but all objects that are used on the whole site. They can be selected, such that their XML representation can be displayed by GUILayoutPanel Additionally the entries in the tree are drawn in three different colours, depending on the state of their templates.

4.2.7 ch.ethz.globis.cmserver.gui.tests

The last package that corresponds to a panel in CMSGui is the tests-package. This package is supposed to be extended with more tests.

GUITestsPanel.java	The tests panel displays a tree with several tests that can be invoked by the user by clicking on the test. A tree was chosen as data structure to allow dynamic grouping of the tests. After clicking on a test, the corresponding information is loaded into the right side of the panel.
GUITestsInterface.java	All the tests have to implement the abstract interface GUITestsInterface. It requires three methods: • toString(): A method for the listing of the test in the tree • getTestPanel(): A panel, where the test is shortly described and where the user can be asked for input values. • OnStart(): The method that is invoked when the user clicks the "Start"-button. The results of the test can then be displayed in a given area, and at the end a value depending on the test returning an error is returned.

	New tests can be created by extending the GUITestsInterface. It is recommended, that tests are saved into this package and that they are named in the following format: <name>Test.java (e.g. StructureTest.java), such that they are easily recognizable. After creating a test, it has to be added to the tree in GUITestsPanel.

4.2.8 ch.ethz.globis.cmserver.gui.images

The images package holds all the icons that are used in CMSGui. This includes some icons on top of the panels and icons within the trees for distinguishing the object-types.

ImageLoader.java	ImageLoader can be used to easily load these images. It looks in its own package for the required image and returns an Icon-object. This has been implemented this way to be able to access the images with a relative path. Therefore, no configuration file is needed for CMSGui.

4.2.9 ch.ethz.globis.cmserver.gui.exception

The last package holds the exception classes used by CMSGui.

GUIException.java	The only exception class used in CMSGui. Usually, errors in CMSGui are handled as soon as possible, such that not too many exceptions are needed. GUIException is a very basic implementation for such errors.

4.3 Classes

To simplify the understanding of the whole program and how the classes work together, the most important classes will now be described in this chapter.

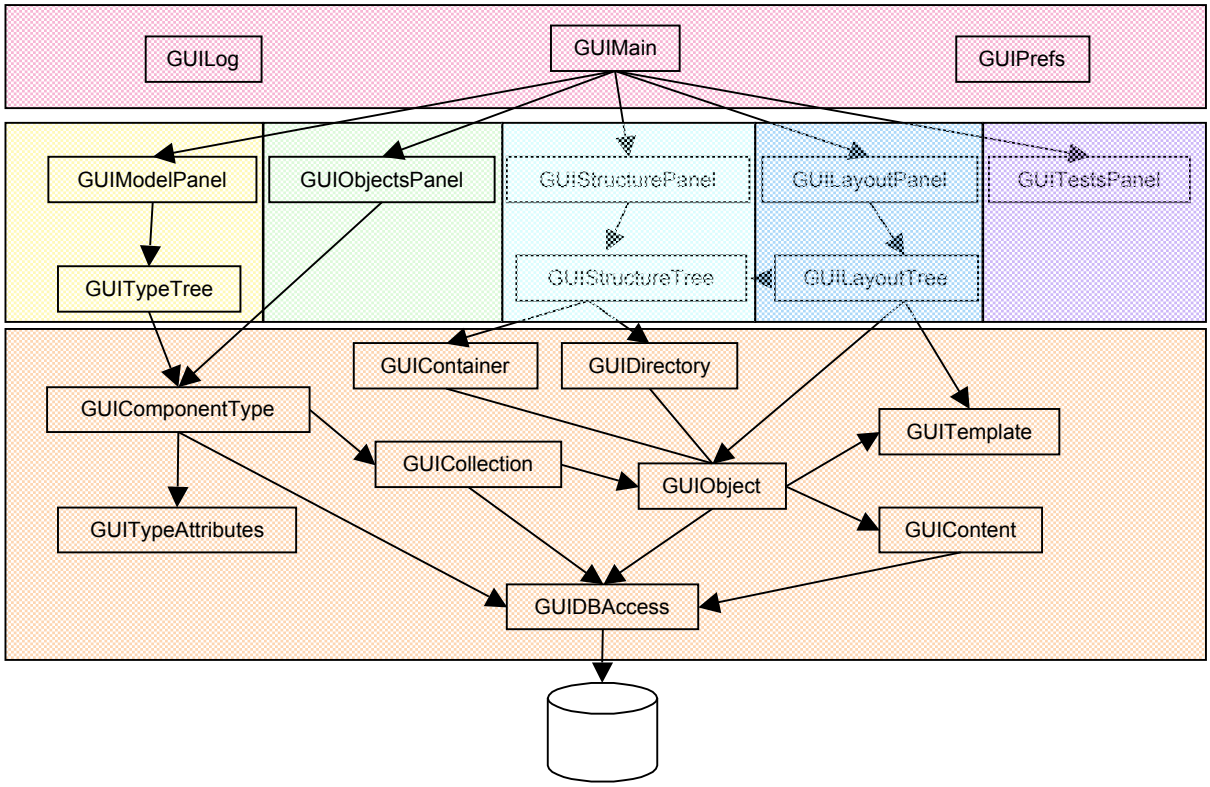

Figure 1: Dependencies of Classes

In Figure 1 the most important classes are brought into relation to each other. The coloured boxes represent the packages described in the previous chapter. The topmost box displays the basic components of the GUI like the windows, menubar, etc…

The five boxes below correspond to the panels that are part of the main window. An arrow in the graphic signifies a relation like “uses” or “consists of”.

The bottommost box is the database-package with the single entry point to the OMS database. All the other classes in this package implement both an API to access and mutate the database object and a GUI to correctly display the object in CMSGui.

In the following sections some of these classes will be explained a little bit more detailed.

4.3.1 GUICollection

GUICollection is the representation of a collection in CMSGui. Each collection is of a particular type and contains a list of objects of this type. GUICollection supports methods like add(object) or remove(object) to manage this list.

As stated before, there may exist several collections of one single type. To simplify the management of such collections, there is one restriction: The structure of the collections of

one type has to be as follows: Every collection is either a leaf collection in which objects can be stored, or it is a parent collection, which is partitioned over its subcollections. The result of this restriction is that each object can be assigned to exactly one leaf collection and that it has to be added automatically to all the parent collections.

To understand this a little bit better, let's have a look at an example. Assume that all collections shown in Figure 2 are of the same type. The following rules can be derived:

- New objects can just be added to collections C, D, E or F. They are then implicitly added to the parent collections.
- All Objects that are in collection E and F are also in collection B.
- Collection A holds all objects that are of this type.

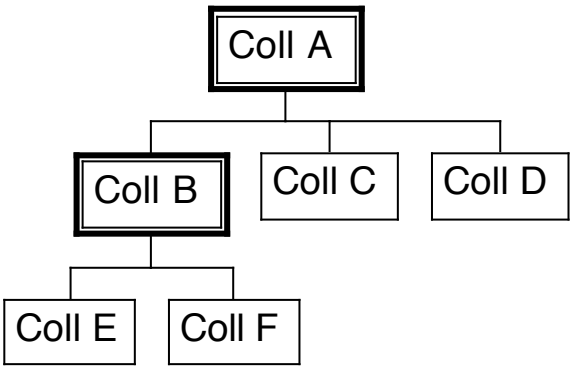

Figure 2: Collection hierarchy

Every GUICollection knows if it is partitioned or not. Additionally to this information, the parent collection is always also known. Every type has exactly one main collection, which is used to access all objects of a certain type.

To make sure, that throughout the whole program the same collections are reused, GUICollection doesn't have a public constructor. Instead, a new GUICollection-instance has to be created using the static method getGUICollection(). This method first checks, if this collection has not already been instantiated. If it hasn't, a new instance is created, otherwise the existing one is returned.

One method of GUICollection is called "CreateNewCollection()". Obviously a new collection in the database can be created this way. It is then automatically added to the partition of the main collection of the same type.

When a collection is removed from the database by using the delete() method, the objects within the collection are not deleted. They remain in the database, because they could still exist in another collection. Therefore when a collection is deleted the contained objects have to be handled before (e.g. moved to another collection or deleted). In CMSGui 1.0 this is not necessary, because collections are only deleted when its type is deleted. In this case all objects have to be removed too!

4.3.2 GUIComponentType

Like many objects in CMSGui, GUIComponentType provides both a Graphical User Interface for its object (in this case a type object), and an API to edit the object.

Part of the user interface is the EditorPanel, which displays all properties of the type, and a method saveChanges() that goes through the EditorPanel and saves the actual values.

Some ComponentTypes are part of the basics of XWCM and cannot be changed or even deleted. Their names all start with the two letters "cm". GUIComponentType doesn't allow any changes to these types. Also there is no way to create new types that start with these letters in CMSGui.

Because the types are mostly used in a tree-context, GUIComponentType extends javax.swing.tree.DefaultMutableTreeNode. The whole tree is built when they are loaded from the database and from then on, each type knows about its parent or its descendants.

The main parts of the API are methods to add or remove attributes or to collect all objects of a certain type. This last functionality is used throughout the program, because whenever an object is needed, the webmaster gets a list of possible objects (e.g.: for an attribute that holds a cmlink, the webmaster gets a drop-down menu with all existing cmlink-objects where he can choose one). There is also a delete-function, which deletes the type and all collections and objects of this type from the database.

Like GUICollection, the constructor of GUIComponentType is protected. New types can just be created by using the static method getGUIComponentType() which looks up the given type. If it does not exist, a new one is created and added to the list.

A lot of getter-methods like “getCollections()” or “getMainCollection()” use private variables to access the wanted result. For example all the collections of a type are stored in a Vector, and whenever they are needed, this Vector is returned. However, new collections may be created without this type knowing about it. For this case, GUIComponentType (and a lot of other classes) has a reload()-method which does nothing else than remove the stored values. When the collections are accessed the next time, they will again be looked up in the database and stored in the Vector.

Beside the collections and attributes, a ComponentType also provides information about some special associations: the content associations! Each object can have some language-dependent content. This content is saved in new objects, the content-objects. The relation between the object and the content is saved in the content-associations. GUIComponentType retrieves what associations can be set for what type.

4.3.3 GUIObject

GUIObject is the representation of a “cmcomponent” object. The functionality is very similar to the one of GUIComponentType. The most important attributes of an object are its type and its collection. In the collection-attribute only the most specific collection is saved. Let’s use the example of chapter 4.3.1 again. If an object is in collection F, it’s also added to collections B and A. The GUIObject just saves a reference to collection F, because from there the other collections can easily be found.

Some other attributes are the templates and contents that are associated to the object. The templates are stored in two hashtables. One stores the default templates of the object, the other one contains the templates that depend on the context of the object. The content objects are also stored in a hashtable. Because in the user interface only one language-content can be viewed at a time, the actually shown contents are additionally saved in another hashtable. The keys of these two hashtables are the associations between object and content (e.g. “TitleContent” or “PictureContent”). Adding and removing of contents is done through GUIObject.

GUIObject’s constructor is also protected. Therefore new instances have to be created with the static method getGUIObject() which assures, that there is not more than one instance for the same object.

To simplify the selection of objects, they have to be displayed in alphabetical order quite often. Therefore GUIObject implements the “Comparable”-interface, which can be used for the sorting of objects.

Like GUIComponentType an EditorPanel manages displaying and editing of all the attribute values. Part of the Panel are also the content-objects. With the button “Save Changes” the newly entered values are stored in the database.

One special method (“getAttributeObjects()”) goes through all the attributes of the object and returns a list of the ones that are themselves GUIObjects. This method is needed for the creation of the LayoutTree that includes all objects used.

Again, a reload()-method assures that changes in other objects which could affect this object are reflected. In this case, the object is simply reloaded from the database.

4.3.4 GUIDirectory

GUIDirectory is an extension of GUIObject and implements a very simple user interface of a directory. To create a GUIDirectory-instance the static method getGUIObject() of GUIObject has to be used. That method distinguishes GUIDirectories, GUIContainers (another extension of GUIObject) and other GUIObjects and creates the appropriate instance.

The EditorPanel of a directory is very similar to the one of the other GUIObjects. The only difference is that one attribute (“root”) is not displayed, because it is not desired that the user may change this attribute. To make sure that there is exactly one directory figuring as root CMSGui sets this attribute automatically. If several directories with this attribute are found when a database is opened, the user is asked to choose one. The other directories are changed to not being root. If none exists, a new directory is created.

A directory has subdirectories and pages, which are stored in special associations. The methods getSubDirectories() and getPages() return lists of the requested GUI-objects. These lists are also loaded just once. If the structure changes, the objects have to be reloaded. In this case, the preloaded lists are deleted, and they will be reloaded the next time they are used.

4.3.5 GUIContent

Each content-object has to be associated to exactly one language. This is not done by the database but has to be ensured by the website-editor. This is done in GUIContent, which holds a content-instance and the asscociated language-object of type GUILanguage. When a new content is created, several properties have to be given: the associated object, the correct language and the association between the object and the content. This last property is also used to determine the type of the content-object.

GUIContent also extends GUIEditableObject, as GUIObject does. This is done to provide the functionality for saving the changes to the attribute values into the database. However, GUIContent does not create its own EditorPanel. It just provides the functionality to collect all attributes and to change their values, but they have to be placed on the EditorPanel of the GUIObject. There are two main reasons, why this was done like this:

- The alignment of the attributes of GUIObject and of GUIContent needed them to be placed on the same panel and not on nested panels.
- To avoid a repainting of the whole panel, whenever a new GUIContent is chosen, all contents of one object are drawn in the same components. This is done with the methods getAttributes(), fillAttributes() and emptyAttributes().

4.3.6 GUITestsPanel

As explained before, the main idea of the tests-panel was that new additional tests could be included later on, based on the requirements of the user.

Therefore an interface was created which defines what methods a new test has to implement. In order to add the test to the tree on the left side of the Tests Panel, a new line has to be inserted in GUITestsPanel.java: In the method createTestTree() new nodes can be created and added to the tree.

The interface of these "tests" is very general and therefore they can also be used to do other things than just testing. For example they can also be used to do some cleanup or for reports.

4.3.7 GUIPrefs

The Preferences of the program are stored in a file called "CMSGUI.pref". This file is saved in the directory, where the program was started. It is read when the program is started, because it contains information like the recently opened files, which is needed when creating the menubar.

Apart some methods to change the preferences from within the program (like adding a new recently opened file) a graphical user interface let's the user choose his preferred settings. Whenever he changes the settings they are saved into the file right away.

Some changes in the preferences may affect existing panels. To make sure, that the program always reflects the latest settings a special hidden button is included in the GUIPreferences. This button is only used to collect different ActionListeners, which can be added by other objects. When the "Save"-button is clicked, the changes are saved and then the hidden button is virtually clicked and thus invokes the registered ActionListeners. For example one part of the Preferences lets the user choose the actual markup language (e.g. HTML or WML) and depending on the chosen markup, the Layout Panel lists different templates.

For more information about the implemented classes read the javadoc-API [5].

5 Conclusions

After having discussed this first implementation of CMSGui we will give a short overview of some missing features or possible improvements in the future.

In the first section (chapter 5.1) we will have a look at the requirements presented earlier and we will explain how they were met.

The following chapters will then describe what still can be done. It can be easily seen that, although the first implemented version of CMSGui works properly, a lot of improvements still can be made.

Chapter 5.4 summarizes, what are the minimal requirements of an OMS-database such that it can be opened in CMSGui.

5.1 Fulfillment of Requirements

In chapter 2 a distinction was made between general requirements to a GUI and the specific requirements to this application. We will now give an overview if and how these requirements could be met in CMSGui 1.0.

Look & Feel

By using Java Swing, a very easy and common way was chosen for the graphical user interface. Everyone can use CMSGui with the Look & Feel he prefers. Buttons, menus, dialogs, dropdown-menus, progressbars, etc… simplify the understanding of what is going on and how the program is to be used.

Usability

The separation of the main tasks into five panels, which can be easily accessed, results in a small GUI that is not overloaded. All functions can be accessed within one or two clicks and are structured in a way that best supports the webmaster's workflow.

Model-Generation

In favour of an easy, intuitive program, some limitations had to be applied to the management of the database model. It is not possible to create several collections or to define associations. Furthermore only a small part of the whole model is displayed, because the other parts cannot be extended by the webmaster.

The result of these restrictions is a user interface that now hides perfectly the underlying schema as it was required.

Object-Creation

The main goal of the objects-part was to have an easy way to create new objects. One single click now creates the object, adds it to the correct collection and displays all corresponding language-dependant contents.

Editing, adding or removing content can also be done with few mouse-clicks.

Site Structure

The fourth panel of CMSGui, the Structure Panel, meets exactly the needs of the required site structure. All directories, pages and other containers can be viewed, edited, added or removed in a simple way. To simplify the user interface, editing of these objects is done in the same way as it is done for all other objects.

Presentation

In CMSGui the XML description of each object is shown when an object is chosen in the Layout Panel. Unfortunately the creation of the XML is not very fast. Especially when a directory or container is selected, a lot of objects have to be traversed.

The selection of templates is very easy and results again in a big advantage compared to usual database-tools.

Just previewing objects does not work fully satisfactorily yet. There is an opportunity to preview objects, but the result sometimes contains undesired characters.

Compatibility

By using as few information about the database model as necessary a high grade of compatibility with future versions was accomplished. The knowledge used also represents the minimal requirements to the database and its model.

5.2 Missing Features

Some features of a content management system have not been implemented in CMSGui. This has two reasons:

- Some functionality is not implemented in CMServer. For example in an OMS database new associations between existing types can be easily created. In CMSGui this is not possible, as no information about associations is used in CMServer. The only way to use associations is by implementing appropriate Java classes. To keep the GUI simple, it made no sense to include functionality that would not be used or that required programming knowledge.
- Other functions have been implemented in the existing CMServer but will be changed drastically in the near future. This includes the whole workflow-management with workflow-states and gatekeepers.

5.3 Future Work

Although CMSGui already can be used to create or edit websites, there are some remaining problems that should be solved to make this program really useful.

Export function

OMS Java mainly works with so-called dumpfiles, which are serialized Java objects. Unfortunately these dumpfiles might change when new versions of OMS Java are used. That means that previously saved files might not be opened by a new version. Although it is possible to import DML and DDL files, there is no way to export the database to such formats later on.

This is definitely one of the most important features that OMS Java has to implement in the near future.

Preview function

As it was explained in chapter 5.1 the preview function does not yet work properly. Besides the unwanted characters, some more problems have to be solved:

- Navigation- and Sitemap-objects need to know about the whole site structure. This means, that they need a special implementation.
- HTML–previews could be loaded automatically into a browser to see the final "product"

Speed

In a next release, some time should be invested in the speed of the tool. Both OMS Java and CMSGui are not really fast. The result is that some operations take too long and block other user interaction.

Workflow

It was explained above, why the workflow was not implemented in CMSGui 1.0. However the workflow is a main feature of content management systems. To make CMSGui more interesting for potential clients, the workflow should be included in CMSGui. One part of it, the user management is supposed to be implemented in OMS Java.

5.4 Minimal database-requirements

As mentioned earlier (see chapter 2.3.5) CMSGui imposes a few restrictions to the databases that can be used. Still, there are some basic requirements that cannot be neglected.

The easiest way to make a database work in CMSGui is to open an existing database in CMSGui and remove all objects that are not needed. Doing so nothing that is still needed can be deleted. In the end all the basic types will remain with their attributes and associations. These are, roughly said, the minimal requirements to a database.

Because some features have not been implemented in CMSGui one might be tempted to delete types such as "CMGatekeeper", etc… Although they are not needed for CMSGui to work properly, they will be used within CMServer when the site is displayed in a browser.

The same rule is also applicable to most of the component-types. To make CMSGui work, at least the following types are needed:

- cmcomponent – basic type for all others
- cmdirectory – special implementation because of site structure (subdirectories and pages)
- cmcontainer – special implementation because of association “hasComponents”

All other types are not necessary, but if one wants to profit of some special behaviour like navigation-objects or sitemaps that automatically scan all underlying directories and pages, one will also need the other types.

Acknowledgements

During the work on my diploma thesis I got a lot of support of different persons. I would like to thank everybody.

- Prof. Moira Norrie
  Thanks for giving me the opportunity to do my diploma thesis in your group. The last four months were very interesting.

- Michael Grossniklaus
  Developer of XWCM and therefore only person who knew how it works…
  He supported me wherever he could and his interest in my work was a big motivation.

- Adrian Kobler
  Developer of OMS Java. Despite a lot of bug-reports I sent to him, he never gave up and always reacted very fast. I'm sure OMS-Java also evolved a lot in the meantime.

- Christoph Duijts
  Thanks for the 24-hour online support and test reading of my report!

Bibliography

[1] M. Grossniklaus. CMServer – An Object-Oriented Framework for Website Development and Content Management, Diploma Thesis, Institute for Information Systems, ETH Zurich, March 3rd 2001

[2] J. Surveyer. Web Tools Review
http://home.inforamp.net/~jbsurv/wb2000.htm

[3] Research Group for Global Information Systems at ETH Zurich
http://www.globis.ethz.ch

[4] S. Zweifel. CMSGui Manual 1.0
The installation and user guide to CMSGui, Version 1.0

[5] S. Zweifel. CMSGUI API
Javadoc-generated API of CMSGUI, Version 1.0